Project Summary

Batch details	PGPDSE-FT GURGAON AUG22
Team members	1. Rishu Gupta 2. Muskan Jain 3. Braj Bhooshan Uraon 4. Ritika Joshi 5. Khushboo Verma
Domain of Project	Sales Analytics
Proposed Project Title	Price prediction of used cars
Group Number	03
Team Leader	Rishu Gupta
Mentor Name	Mr Jayveer Nanda

Date : 23rd Nov 2022

Signature of Mentor

Team Leader

Table of Content

Project Details

Overview

Starting its journey from the day when the first car rolled on the streets of Mumbai in 1898, the Indian automobile industry has demonstrated a phenomenal growth to this day. Today Indian automobile industry present a wide range of varieties and models meeting all the globally established standards. This industry accounts for ~7 percent of country's GDP in Financial Year 2015-16.

Dataset Name:

Used Cars Dataset

Business Problem Statement

1. Business Problem Understanding

A used car, a pre-owned vehicle, or a second hand car, is a vehicle that has previously had one or more retail owners. Used cars are sold through a variety of outlets, including franchise and independent car dealers, rental car companies Currently the pre-owned car market size is estimated to be at 3 million in India and it is growing. Indian pre-owned market is dominated by the unorganized players to an extent of 88 percentage. This comprise 80 percent of the deals owner to owner transaction out of which 30 percent through brokerage mediated by the individuals. A miniscule 12 percent of the pre-owned cars market is controlled by organized sector. The areas that worries pre owned car purchaser is Authenticity of the car, availability of the spare parts, *the optimal price of the used car.*

2. Business Objective

The main objective of our work is to build a Machine Learning model, which predicts the optimal price of a pre-owned car or used car on the basics its features for the business. Which brings the price transparency between the buyer and seller in the used car ecosystem. In an unorganized ecosystem the price of a pre-owned vehicle boils down to awareness, negotiation skills, availability of vehicle, price sensitivity thus, we are trying to solve for the inefficiencies in the used car ecosystem by leveraging data analytics.

3. Limitation of the Project

Our Model does not incorporate very advance deep learning and reinforcement learning algorithms due to computational power reason.

Variables Detail

- Independent variables:

  id, url, region, region_url, year, manufacturer,

  model, condition, cylinders, fuel, odometer, title_status,

  transmission, VIN, drive, size, type, paint_color,

  image_url, description, county, state, lat, long, posting_date

- Target Variable

  Price

Variable information/Data description

id	entry ID
url	listing URL
region	craigslist region
region_url	region URL
price	entry price
year	entry year

manufacturer	manufacturer of vehicle
model	model of vehicle
condition	condition of vehicle
cylinders	number of cylinders
fuel	fuel type
odometer	miles travelled by vehicle
title_status	title status of vehicle
transmission	transmission of vehicle
VIN	vehicle identification number
drive	type of drive
size	size of vehicle
type	generic type of vehicle
paint_color	color of vehicle
image_url	image URL
description	listed description of vehicle
county	useless column left in by mistake
state	state of listing
lat	latitude of listing
long	longitude of listing

posting_date	posting date

Data Preparation

We need to import following libraries for the working on the dataset

```
# import 'Pandas'
import pandas as pd

# import 'Numpy'
import numpy as np

# import subpackage of Matplotlib
import matplotlib.pyplot as plt

# import 'Seaborn'
import seaborn as sns

# to suppress warnings
from warnings import filterwarnings
filterwarnings('ignore')

# display all columns of the dataframe
pd.set_option('Display.max_columns',None)

# display all rows of the dataframe
pd.set_option('Display.max_rows',None)

# to display the float values upto 6 decimal places
pd.options.display.float_format = '{:.2f}'.format

# import train-test split
from sklearn.model_selection import train_test_split

# import various functions from statsmodels
import statsmodels
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import power_transform
from sklearn.preprocessing import OrdinalEncoder
```

```
# import various functions from statsmodel to perform linear regression
import statsmodels
import statsmodels.api as sm
import statsmodels.stats.api as sms

# import function to perform GridSearchCV
from sklearn.model_selection import GridSearchCV

# import functions to perform cross validation
from sklearn.model_selection import KFold

#importing function for DecisionTreeRegressor
from sklearn.tree import DecisionTreeRegressor

#importing function for RandomForestRegressor
from sklearn.ensemble import RandomForestRegressor

#importing function for AdaBoostRegressor,BaggingRegressor,GradientBoostingRegressor,
#StackingRegressor
from sklearn.ensemble import AdaBoostRegressor,BaggingRegressor
from sklearn.ensemble import StackingRegressor,GradientBoostingRegressor
```

```
# import 'stats'
import scipy.stats as stats

# for normality Test
from scipy.stats import shapiro,mannwhitneyu

# 'metrics' from sklearn is used for evaluating the model performance
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score


# import function to perform linear regression
from sklearn.linear_model import LinearRegression

# import StandardScaler to perform scaling
from sklearn.preprocessing import StandardScaler

# import SGDRegressor from sklearn to perform linear regression with stochastic gradient descent
from sklearn.linear_model import SGDRegressor

# import function for ridge regression
from sklearn.linear_model import Ridge

# import function for lasso regression
from sklearn.linear_model import Lasso

# import function for elastic net regression
from sklearn.linear_model import ElasticNet

# import function to perform GridSearchCV
from sklearn.model_selection import GridSearchCV

# import functions to perform feature selection
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
from sklearn.feature_selection import RFE
```

Here we have read the csv file using pandas function:

```
: #loading the dataset
df=pd.read_csv('vehicles.csv')
```

Before we start modelling our data, we need to modify our data as per inputs required for Machine learning algorithms. In order to achieve it, we need to check and modify the data.

Following shows the statistical description of the Numerical data in the original Dataset

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
id	426880.00	7311486634.22	4473170.41	7207408119.00	7308143339.25	7312620821.00	7315253543.50	7317101084.00
price	426880.00	75199.03	12182282.17	0.00	5900.00	13950.00	26485.75	3736928711.00
year	425675.00	2011.24	9.45	1900.00	2008.00	2013.00	2017.00	2022.00
odometer	422480.00	98043.33	213881.50	0.00	37704.00	85548.00	133542.50	10000000.00
county	0.00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
lat	420331.00	38.49	5.84	-84.12	34.60	39.15	42.40	82.39
long	420331.00	-94.75	18.37	-159.83	-111.94	-88.43	-80.83	173.89

From the above description we can see that in our Original Dataset we have extreme values in our target variable (Price), Price max value is 3736928711.00 which is not possible. Since it is our Target variable we can not treat outliers so we decided to trim to 10 percentile data from top and bottom values.

```
df=df1[43001:384000]
```

Data Information

It contains the number of rows and columns in the DataFrame

```
df.shape
```

```
(340999, 26)
```

Following shows the statistical description of the Numerical data

```
# five point summary of data
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
id	340999.00	7311562525.94	4455911.05	7207408119.00	7308238494.00	7312769304.00	7315272978.50	7317101084.00
year	340373.00	2010.36	9.48	1900.00	2007.00	2013.00	2016.00	2022.00
county	0.00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
odometer	339110.00	104822.64	198363.38	0.00	46000.00	95356.00	141000.00	10000000.00
lat	337861.00	38.48	5.77	-81.84	34.78	39.24	42.30	82.39
long	337861.00	-93.86	17.94	-159.72	-106.88	-87.71	-80.66	167.63
price	340999.00	15590.58	9932.34	500.00	6995.00	13950.00	23000.00	37500.00

Following shows the statistical description of the object data

```
# describation of object variable
df.describe(include="object").T
```

	count	unique	top	freq
url	340999	340999	https://bham.craigslist.org/cto/d/pelham-2020-...	1
region	340999	404	columbus	3082
region_url	340999	413	https://milwaukee.craigslist.org	2794
manufacturer	328090	42	ford	54172
model	337172	24952	f-150	5811
condition	216757	6	good	105208
cylinders	204125	8	6 cylinders	78100
fuel	338691	5	gas	295209
title_status	334651	6	clean	322075
transmission	339410	3	automatic	267308
VIN	201426	92403	1FMJU1JT1HEA52352	261
drive	236237	3	4wd	94804
size	101586	4	full-size	51687
type	266095	13	sedan	74938
paint_color	242141	12	white	61092
image_url	340937	203205	https://images.craigslist.org/00N0N_1xMPvfxRAI...	5966
description	340936	292192	Call or text today to find out more. (602) 620...	184
state	340999	51	ca	39485
posting_date	340937	309702	2021-04-13T13:19:15-0500	11

From the above data we can see that there are initially 340999 rows in the Data Frame and 26 columns.

From the five point summary we can see that the max values of price is 37500 USD, while the min is 500 USD where the median value is 13950 USD.

Listed cars are manufactured earlies in 1900, and latest is of 2022 while the 2013 is the median year of manufacturing of listed car.

Odometer reading has max values as 10000000 unints, while the mix in 0, Median odometer reading is 95356 units. From the five point summary we can see that there are outliers present the odometer.

Data Summary

Following is the info of the Data which shows the column name, not-null count and data type of respective column

```
# data info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 340999 entries, 746 to 180025
Data columns (total 26 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   id            340999 non-null  int64  
 1   url           340999 non-null  object 
 2   region        340999 non-null  object 
 3   region_url    340999 non-null  object 
 4   year          340373 non-null  float64
 5   manufacturer  328090 non-null  object 
 6   model         337172 non-null  object 
 7   condition     216757 non-null  object 
 8   cylinders     204125 non-null  object 
 9   fuel          338691 non-null  object 
 10  title_status  334651 non-null  object 
 11  transmission  339410 non-null  object 
 12  VIN           201426 non-null  object 
 13  drive         236237 non-null  object 
 14  size          101586 non-null  object 
 15  type          266095 non-null  object 
 16  paint_color   242141 non-null  object 
 17  image_url     340937 non-null  object 
 18  description   340936 non-null  object 
 19  county        0 non-null       float64
 20  state         340999 non-null  object 
 21  posting_date  340937 non-null  object 
 22  odometer      339110 non-null  float64
 23  lat           337861 non-null  float64
 24  long          337861 non-null  float64
 25  price         340999 non-null  int64  
dtypes: float64(5), int64(2), object(19)
memory usage: 70.2+ MB
```

Data Type Conversion

```
# changing datatype numerical to object
for i in ['year','county']:
    df[i]=df[i].astype(object)
```

In our dataset, year, county are in int type. We have change them into object type, With the help of pandas library and 'astype()' method, we have changed them to object type.

From the above, we can see that data type of year, county changed to object type.

Duplicate Data

```
df.duplicated().value_counts()
```

```
False    340975
True         24
dtype: int64
```

```
df.drop_duplicates(inplace=True)
```

With the help of duplicate function available in python, we checked for duplicate rows. By performing the method, we found that there were 24 duplicates rows in the DataFrame which further we dropped them and form the DataFrame.

Data Redundancy

```
df.drop(['id','county','url','region_url','image_url','description','VIN'],axis=1,inplace=True)
```

```
df.columns
```

```
Index(['region', 'year', 'manufacturer', 'model', 'condition', 'cylinders',
       'fuel', 'title_status', 'transmission', 'drive', 'size', 'type',
       'paint_color', 'state', 'posting_date', 'odometer', 'lat', 'long',
       'price'],
      dtype='object')
```

```
df.shape
```

```
(340999, 19)
```

Here these columns (id, county, url, region_url, image_url, description, VIN) are dropped which is redundant for future analysis.

Missing values

Missing values plays a prominent role in the dataset. Generally, we can drop the columns or rows depending the percentage of missing values. We can also replace the missing values with optimum values. In order to perform such operations, we willfirst look into the overall missing values in each column using the below python code.

```
# view missing value in data
df.isnull().sum()
```

```
id                   0
url                  0
region               0
region_url           0
year               626
manufacturer     12909
model             3827
condition       124242
cylinders       136874
fuel              2308
title_status      6348
transmission      1589
VIN             139573
drive           104762
size            239413
type             74904
paint_color      98858
image_url           62
description         63
county          340999
state                0
posting_date        62
odometer          1889
lat               3138
long              3138
price                0
dtype: int64
```

Missing values in percentage.

	Column_Names	Null Percent
19	county	100.00
14	size	70.21
12	VIN	40.93
8	cylinders	40.14
7	condition	36.43
13	drive	30.72
16	paint_color	28.99
15	type	21.97
5	manufacturer	3.79
10	title_status	1.86
6	model	1.12
23	lat	0.92
24	long	0.92
9	fuel	0.68
22	odometer	0.55
11	transmission	0.47
4	year	0.18
21	posting_date	0.02
17	image_url	0.02
18	description	0.02
0	id	0.00
20	state	0.00
1	url	0.00
3	region_url	0.00
2	region	0.00
25	price	0.00

Treating Missing values

Here we are dropping columns (condition, cylinders,
Drive, transmission) which have high null values and its treatment can affect the whole data.
We drop the rows which has null values in condition, cylinders,
Drive, transmission features using dropna( ) with axis, as we cant drop the whole column as they are important features.

```
df.dropna(subset=['condition','cylinders','transmission','drive'],axis=0,inplace=True)
```

```
df.shape
```

```
(131916, 19)
```

Further in the categorical columnsn in manufacturer, model, title_status, size, type, paint_color where there were null values, we have imputed null values with the mode as mode values of respective columns are those which are highest frequency with in the column.

And in Numerical columns in odometer, null values is imputed with median as median is the middle value of all the values when sorted and it'll not get affected by outliers.

Imputing mode in column manufacturer:

```
df['manufacturer'].value_counts()
```

```
ford                21896
```

```
df.loc[df['manufacturer'].isnull(),'manufacturer'] = 'ford'
```

Imputing mode in column model:

```
df['model'].value_counts()
```

```
f-150                                                  1899
```

```
df.loc[df['model'].isnull(),'model'] = 'f-150'
```

Imputing mode in column title_status:

```
df['title_status'].value_counts()
```

```
clean          122708
rebuilt          4062
salvage          1887
lien              901
missing           342
parts only         70
Name: title_status, dtype: int64
```

```
df.loc[df['title_status'].isnull(),'title_status'] = 'clean'
```

Imputing mode in column size:

```
df['size'].value_counts()
```

```
full-size      43126
mid-size       24174
compact        10442
sub-compact     1521
Name: size, dtype: int64
```

```
df.loc[df['size'].isnull(),'size'] = 'full-size'
```

Imputing mode in column type:

```
df['type'].value_counts()
```

```
sedan          32939
SUV            29260
truck          14660
pickup         12157
coupe           7616
hatchback       4820
convertible     3588
van             3448
other           3200
wagon           3071
mini-van        2950
offroad          505
bus              233
Name: type, dtype: int64
```

```
df.loc[df['type'].isnull(),'type'] = 'sedan'
```

Imputing mode in column paint_color:

```
df['paint_color'].value_counts()
```

```
white     27400
black     20961
silver    17008
blue      12912
grey      12570
red       12006
green      4071
brown      3384
custom     2611
yellow     1126
orange      734
purple      378
Name: paint_color, dtype: int64
```

```
df.loc[df['paint_color'].isnull(),'paint_color'] = 'white'
```

Imputing median in column odometer:

```
df['odometer'].median()
```

```
110212.0
```

```
df.loc[df['odometer'].isnull(),'odometer'] = 110212.0
```

Treating Outliers

As we have only one numerical feature that is odometer, and we have outliers in it but these outliers are good outliers as there could be chances that a car can have an extreme odometer reading. Although we dropped rows where odometer reading was zero.

Univariate Analysis

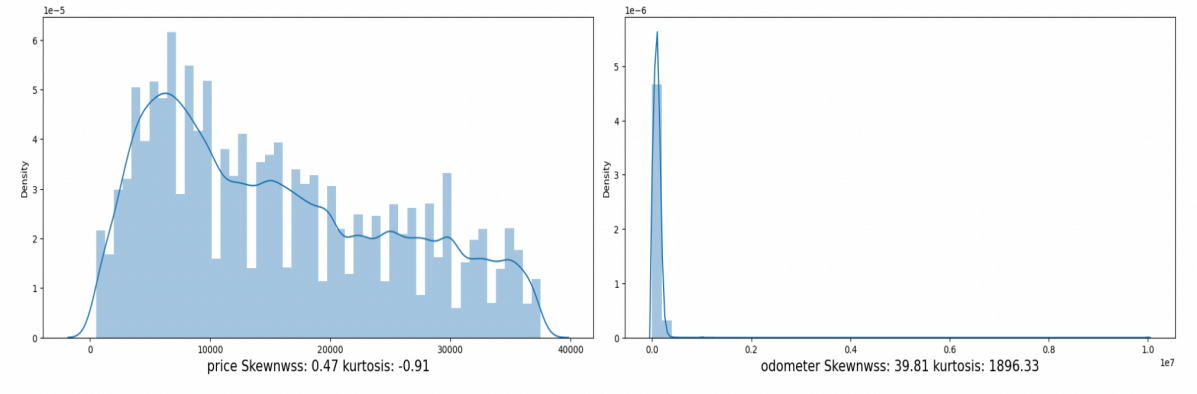
Univariate Analysis is the simplest form of analyzing data, uni- means one so in other words the data has only one variable. Univariate data requires to analyze each variable separately.

Numerical Feature : Odometer, Price

Distplot of numerical features

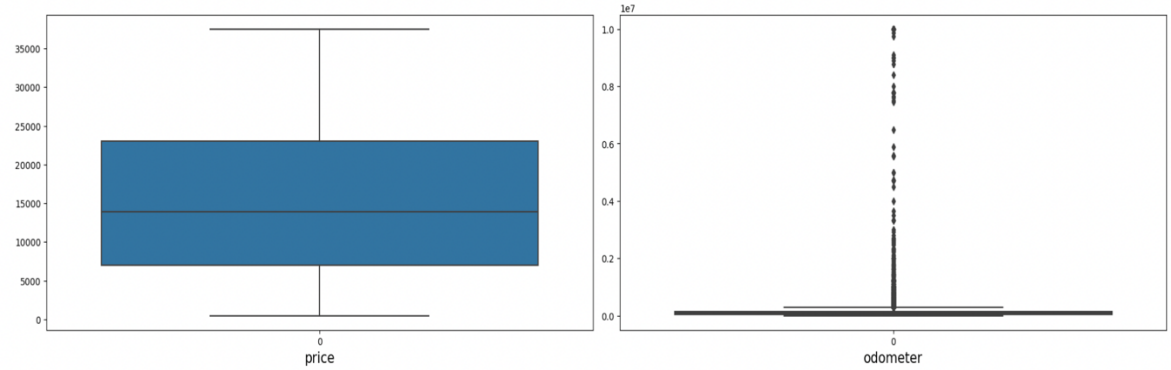
```
# DISTPLOT
cols=['price', 'odometer']
plt.figure(figsize=(20,15))
t=1
for i in cols:
    plt.subplot(3,2,t)
    sns.distplot(df[i])
    plt.xlabel('%s Skewnwss: %.2f kurtosis: %.2f'%(i,df[i].skew(),df[i].kurt()),fontsize=15)

    t+=1
plt.tight_layout()
```



Boxplot of numerical features

```
# BOX PLOT
cols=['price', 'odometer']
plt.figure(figsize=(20,15))
t=1
for i in cols:
    plt.subplot(3,2,t)
    sns.boxplot(df[i])
    plt.xlabel(i,fontsize=15)
    t+=1
plt.tight_layout()
plt.show()
```



From the above plots we can see the distribution of the Price and odometer.

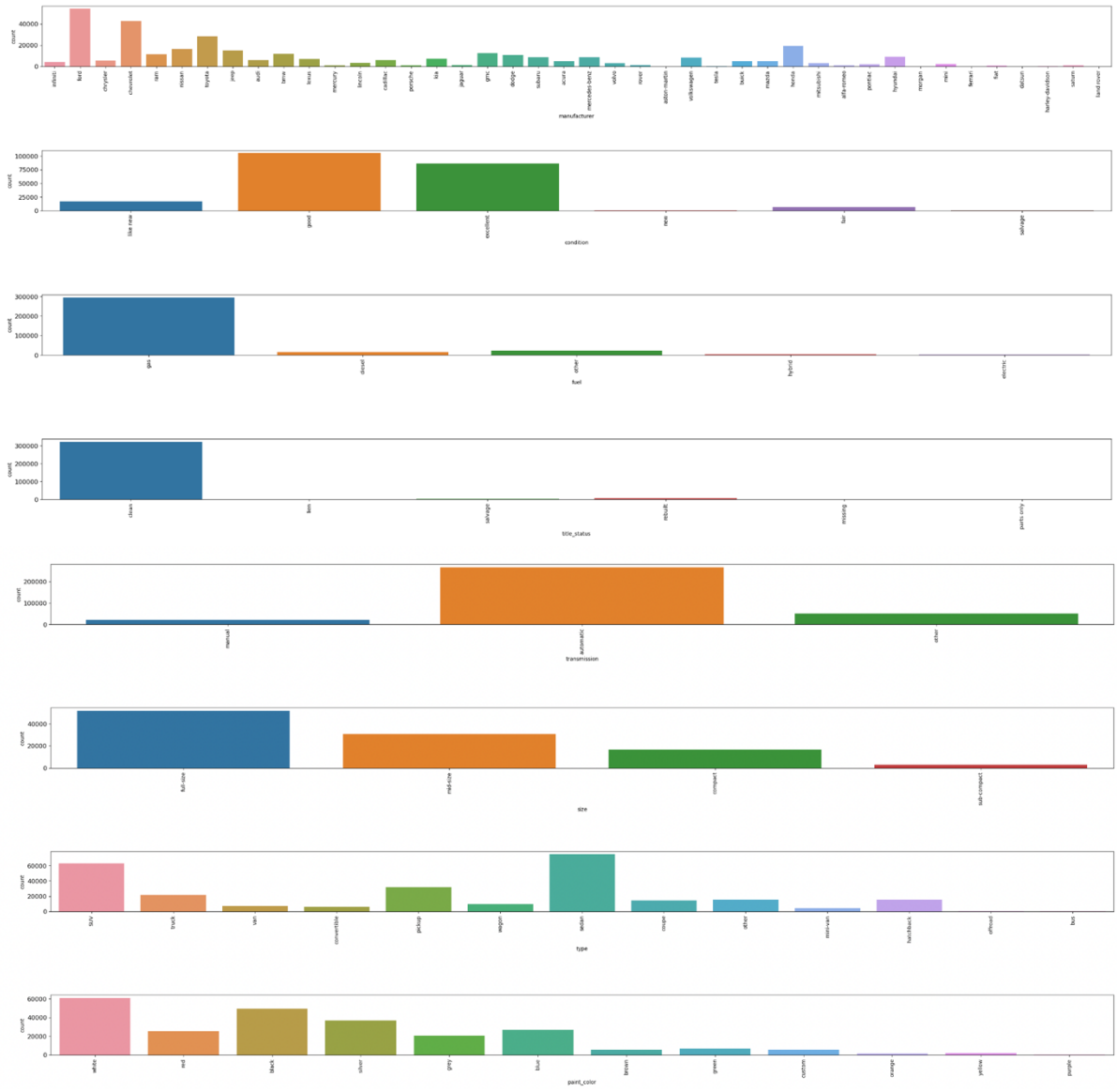
Price is kind of uniformly distributed having skewness 0.47 and kurtosis -0.91

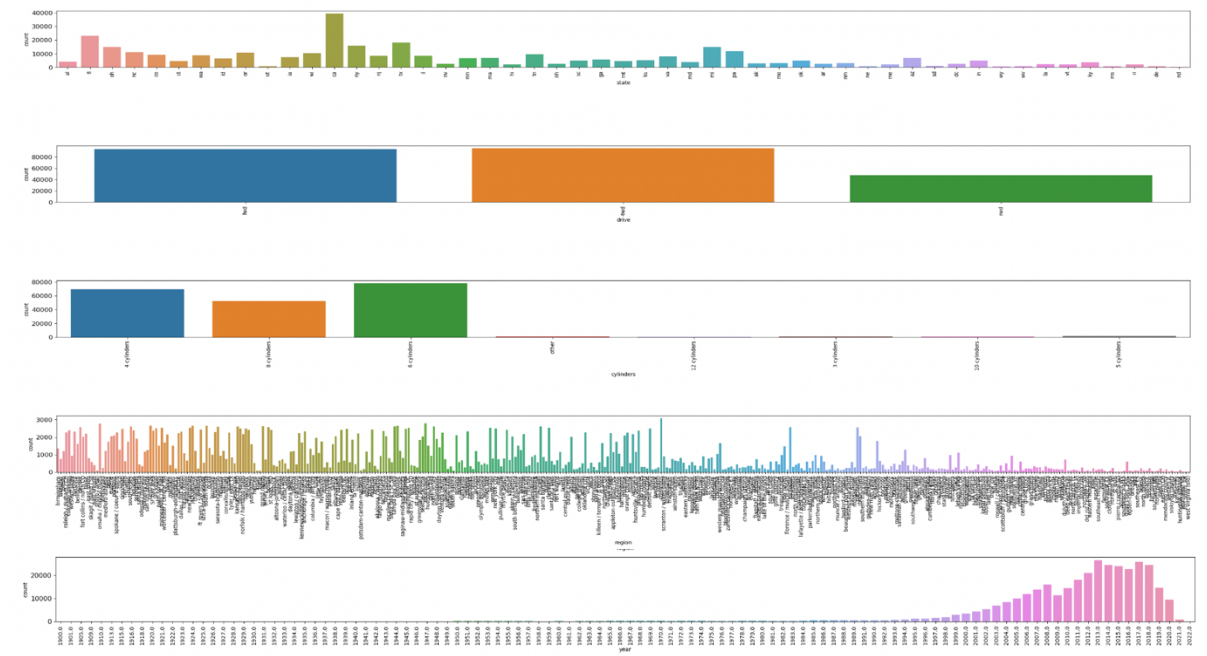
Odometer is highly skewed with skewness of 39.81 and kurtosis 1896.33

There are outliers present in the data in odometer.

Categorical Features: 'manufacturer', 'condition','fuel', 'title_status', 'transmission',
'size','type','paint_color','state','drive','cylinders','region','year'

Count plot of categorical features





The sale of car from FORD is the highest followed by Chevrolet and Toyota,And miniumun sale is from aston-martin,morgan and Ferrari.

The number of Full size cars are more followed by mid-size.

The Number of sedan type cars is most, followed by SUV and pickup. Where the least is bus and off road.

Most number of cars are sold from 2013 manufactured.

The number of cars which runs on gasoline are maximum, whereas electric cars count is least.

Most number of cars for resale are listed from California , While the least number of cars which are listed for resale are from Wyoming.

The max number of cars being sold are of 6,4,8 cylinders.

Most care being sold are of white color , followed by black and silver,While the least sold cars are of purple.

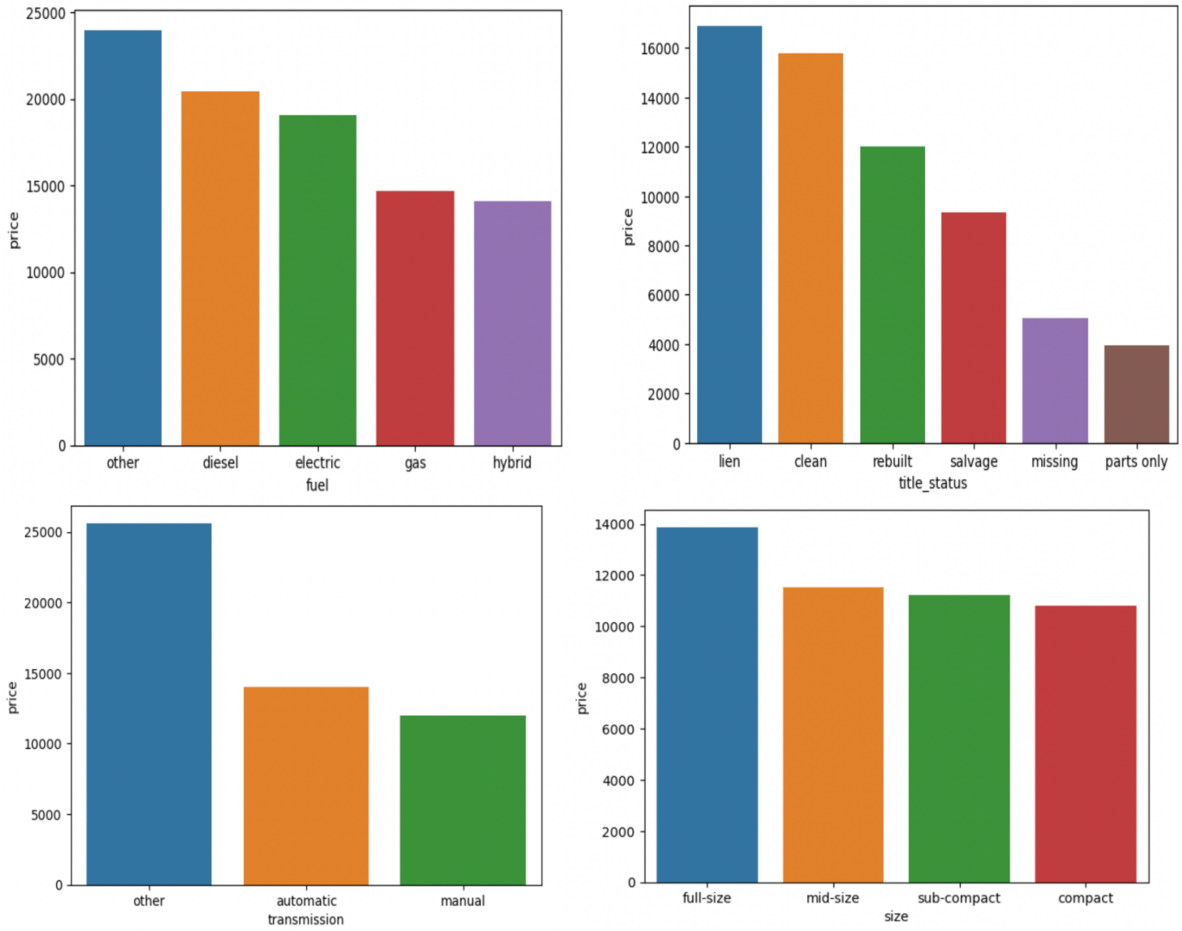
The use of pre-owned cars is in increasing trend.

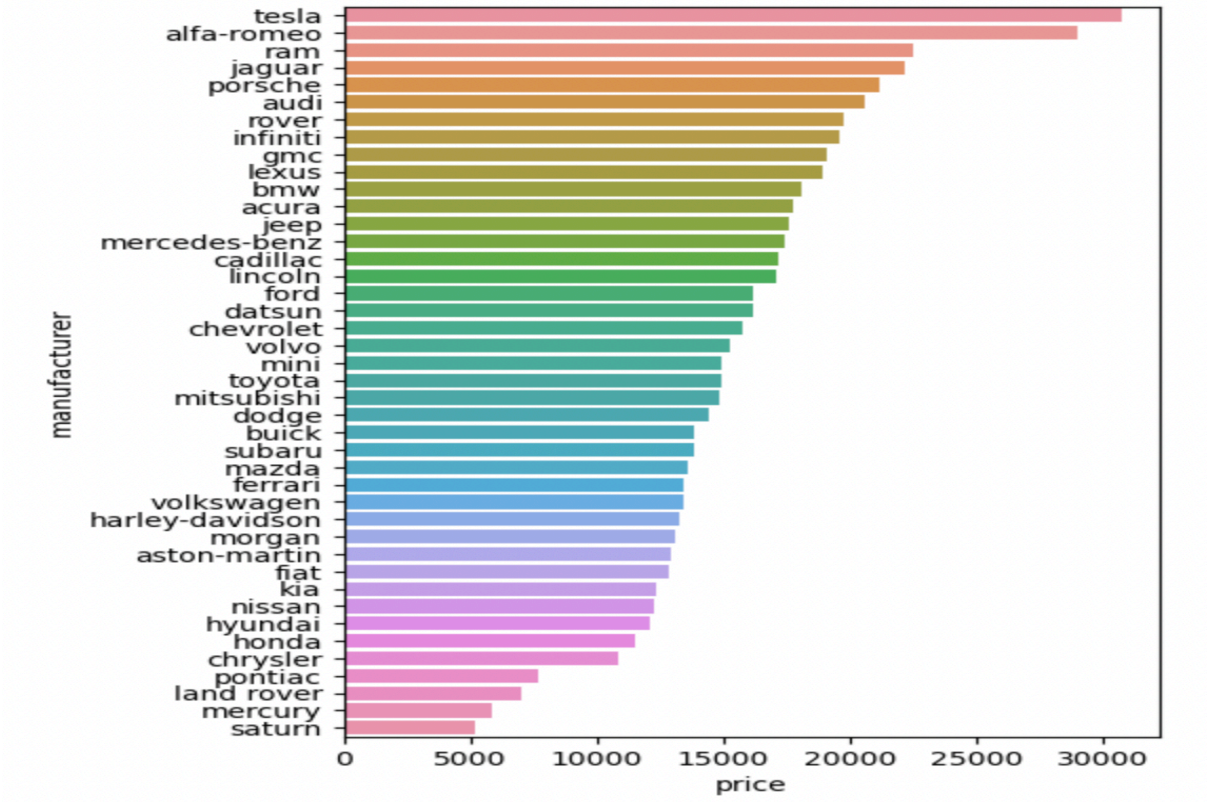
Bivariate Analysis

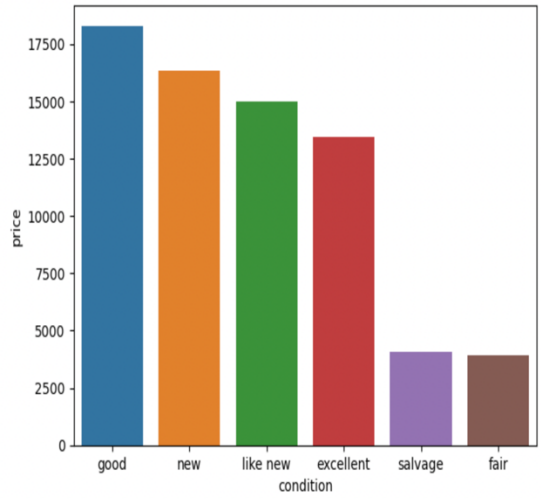
Bivariate Analysis is one of the forms on analysis, which involves the analysis of two quantitative analysis. It involves the analysis of two variables for the purpose of determining the empirical relation between them. In this one variable is dependent and other is independent.

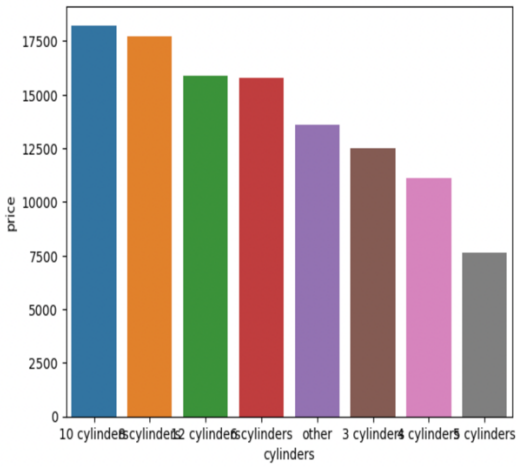
Category Vs Numerical

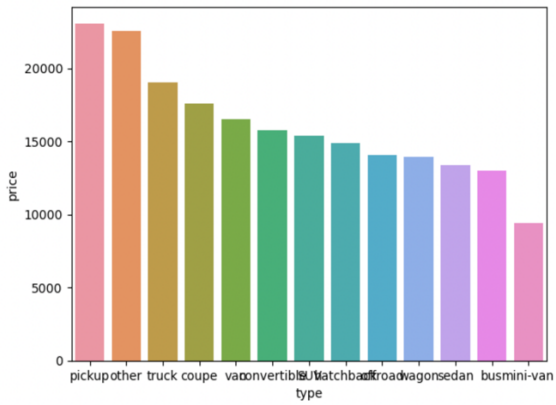
Barplot of different independent features with Avg price (target feture/dependent feature)

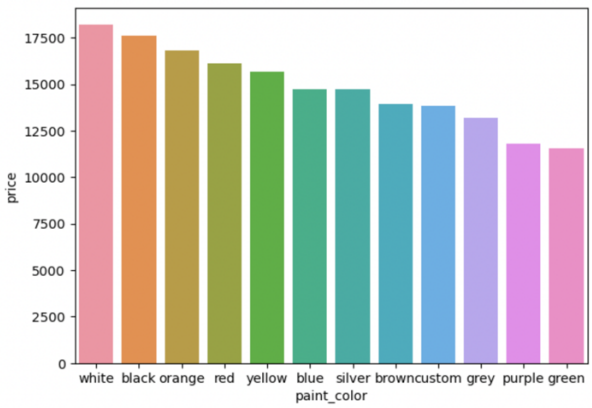



tesla
alfa-romeo
ram
jaguar
porsche
audi
rover
infiniti
gmc
lexus
bmw
acura
jeep
mercedes-benz
cadillac
lincoln
ford
datsun
chevrolet
volvo
mini
toyota
mitsubishi
dodge
buick
subaru
mazda
ferrari
volkswagen
harley-davidson
morgan
aston-martin
fiat
kia
nissan
hyundai
honda
chrysler
pontiac
land rover
mercury
saturn
manufacturer
0
5000
10000
15000
20000
25000
30000
price


price
good
new
like new
excellent
salvage
fair
condition


price
10 cylinders
8 cylinders
12 cylinders
6 cylinders
other
3 cylinders
4 cylinders
5 cylinders
cylinders


price
pickup
other
truck
coupe
van
convertible
SUV
hatchback
offroad
wagon
sedan
bus
mini-van
type


price
white
black
orange
red
yellow
blue
silver
brown
custom
grey
purple
green
paint_color

The Average Price of cars being sold is highest in West Virginia followed Alaska and Alabama, Other States have almost similar Average Price.

Tesla cars have highest average price, followed by alfa- Romeo, While the least average price is of Satum.

Highest Average price is of cars which have 10 cylinders, while the lowest average price is of cars which have 5 cylinders.

The average price of the full size cars is highest while the rest have similar average price.

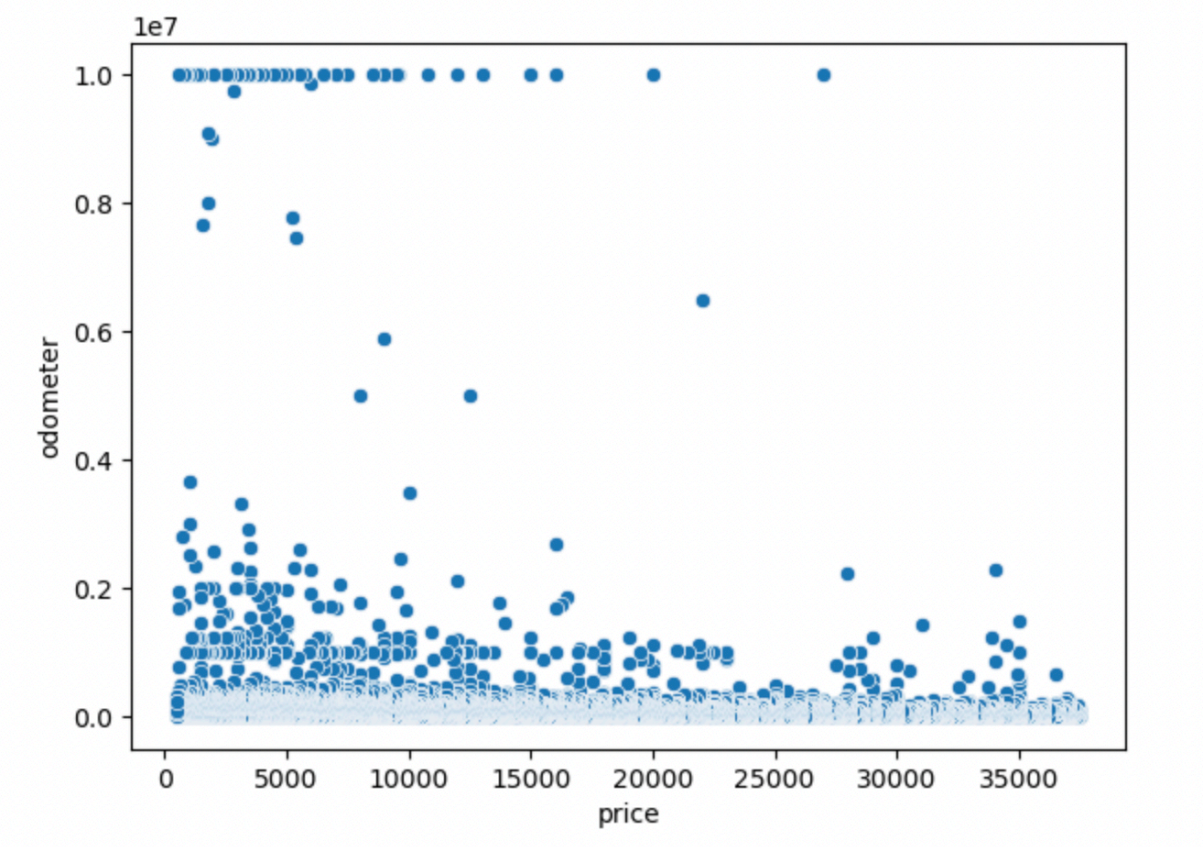
The average price of pickup cars is highest followed by others while the least is of mini-van.

The average highest price is of white color followed by black and orange while the least is of purple and green.

Numerical Vs Numerical

Scatter Plot of Odometer with Price

```
plt.figure(figsize=(7,5))
sns.scatterplot(x=df['price'],y=df['odometer'])
plt.show()
```



Odometer vs Price-Majority of the odometer reading are upto 200000 units.
There are extreme values present in the odometer readings.

Feature Engineering

Extracting a new feature ‘age’ of the Car from the year of manufactured and current year.

```
# EXTRACTING age of the vehicles
df['age'] = 2022-df.year
```

```
df['age'].head()
```

```
746       2.00
101332    5.00
302538    8.00
240731   54.00
84607     5.00
Name: age, dtype: object
```

Classifying the color of the cars in less categories, in light color, dark color and custom.

```
light=['white','yellow','silver','orange']
```

```
dark = ['red', 'black', 'blue', 'green', 'grey','brown','purple']
```

```
def color(c):
    for i in light:
        if str(c) == 'custom':
            return('custom')
        elif str(c)==i:
            return('light')
        else:
            return('dark')
```

```
df2['color'].value_counts()
```

```
dark      85150
light     43788
custom     2611
Name: color, dtype: int64
```

Cleaning the feature cylinders, by removing word ‘cylinders’ form DataFrame.

```
# removing the word cylinder from cylinder column to make it short and easier to encode
df['cylinders'] = df['cylinders'].str.rstrip('cylinders')
```

```
df['cylinders'].head()
```

```
746       4
101332    8
302538    6
240731    8
84607     8
Name: cylinders, dtype: object
```

MODEL BUILDING

LINEAR REGRESSION:

LINEAR REGRESSION USES THE RELATIONSHIP BETWEEN THE DATA-POINTS TO DRAW A STRAIGHT LINE THROUGH ALL THEM. THE CORE IDEA IS TO OBTAIN A LINE THAT BEST FITS THE DATA. THE BEST FIT LINE IS THE ONE FOR WHICH TOTAL PREDICTION ERROR (ALL DATA POINTS) ARE AS SMALL AS POSSIBLE. ERROR IS THE DISTANCE BETWEEN THE POINT TO THE REGRESSION LINE.

DATA PREPROCESSING:

1. Encoding:

```
# encoding
# frequency

v_df['manufacturer'] = v_df['manufacturer'].map(v_df['manufacturer'].value_counts())
v_df['model'] = v_df['model'].map(v_df['model'].value_counts())
v_df['condition'] = v_df['condition'].map(v_df['condition'].value_counts())
v_df['title_status'] = v_df['title_status'].map(v_df['title_status'].value_counts())
v_df['type'] = v_df['type'].map(v_df['type'].value_counts())
v_df['state'] = v_df['state'].map(v_df['state'].value_counts())
```

```
# Ordinal
oe = OrdinalEncoder()
v_df['cylinders'] = oe.fit_transform(np.array(v_df.cylinders).reshape(-1,1))
```

```
# 1-Hot
v_df_new = pd.get_dummies(data = v_df,
                          columns=['fuel','transmission','drive','color'],
                          drop_first=True)
```

2. Transforming and Scaling:

```
# Tansformation
# Importing Power transformer from sklearn library
# By default it will transform the data using yeo-johnson and also scale the data
from sklearn.preprocessing import power_transform
v_df_new[['year','odometer','age']] = power_transform(v_df_new[['year','odometer','age']])
```

3. Splitting Dependent and Independent Variables:

```
# Splitting into X and y.
# X - Independent variables or Predicors
# y - Dependent Variable or Target Variable
X = v_df_new.drop(columns = ['paint_color','price','Date'])
y = v_df_new['price']
```

Train Test Split and Model Building:

we have to split the dataset into train and test data using 'model_selection 'library from scikit. In general, train test split will be done in the ratio of 70:20 and we followed the same for our model building.

1. **Linear Regression:**

Simple linear regression is a statistical method that allows us to summarize and study relationships between two continuous (quantitative) variables.

One variable, denoted x, is regarded as the predictor, explanatory, or independent variable. The other variable, denoted y, is regarded as the response, outcome, or dependent variable.

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modelling including classification, regression,

clustering and dimensionality reduction via a consistence interface in Python.

```
# Creating Train test and Model

Xtrain,Xtest,ytrain,ytest = train_test_split(X,y,test_size=0.20,random_state=0)
lr = LinearRegression() # Initiate Model
lrmodel = lr.fit(Xtrain,ytrain) # Fitting Data in the model
ypred = lrmodel.predict(Xtest) # Predicting the Target variable
print('Train R2',lrmodel.score(Xtrain,ytrain)) # Training R squared
print('Test R2',r2_score(ytest,ypred))# Test R Squared
print('Test RMSE',np.sqrt(mean_squared_error(ytest,ypred))) # RMSE
```

```
Train R2 0.643300886066092
Test R2 0.6431072423573934
Test RMSE 5736.185264263632
```

The Training R squared is 64% meaning that the model is not able to learn properly and our data is not linear enough , that is, it is spread out. The error value is also high.

2. Linear Regression Without scaling:

```
# fitting into model
lr = LinearRegression()
lrmodel = lr.fit(Xtrain,ytrain)
ypred = lrmodel.predict(Xtest)
```

```
# checking R2 score and RMSE
print('Test R2',r2_score(ytest,ypred))
print('Test RMSE',np.sqrt(mean_squared_error(ytest,ypred)))
```

```
Test R2 0.4377879535820184
Test RMSE 7200.008057187531
```

3. Linear Regression With scaling

```
lr = LinearRegression()
lrmodel = lr.fit(Xtrain,ytrain)
ypred = lrmodel.predict(Xtest)
print('Test R2',r2_score(ytest,ypred))
print('Test RMSE',np.sqrt(mean_squared_error(ytest,ypred)))
```

```
Test R2 0.44075766539479566
Test RMSE 7180.493973271001
```

```
# No improvement
```

As can be seen scaling has no effect on the model. Instead the transformed data performed much better.

4. Lasso:

Lasso regression is a type of **linear regression** that uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean. The lasso procedure encourages simple, sparse models (i.e. models with fewer parameters). This particular type of regression is well-suited for models showing high levels of muticollinearity or when you want to automate certain parts of model selection, like variable selection/parameter elimination.

The "LASSO" stands for **L**east **A**bsolute **S**hrinkage and **S**election **O**perator.

```
# Using Lasso Model
ls = Lasso() # Initiate Model
lsmodel = ls.fit(Xtrain,ytrain) # Fitting Data in the model
ypred = lsmodel.predict(Xtest) # Predicting the Target variable
print('Train R2',lsmodel.score(Xtrain,ytrain)) # Training R squared
print('Test R2',r2_score(ytest,ypred)) # Test R Squared
print('Test RMSE',np.sqrt(mean_squared_error(ytest,ypred))) # RMSE
```

```
Train R2 0.6432861640409532
Test R2 0.6430719979725344
Test RMSE 5736.468491263606
```

5. Ridge:

Ridge regression is a way to create a parsimonious model when the number of predictor variables in a set exceeds the number of observations, or when a data set has multicollinearity (correlations between predictor variables).

```
# Using Ridge Model
rd = Ridge()
rdmodel = rd.fit(Xtrain,ytrain)
ypred = rdmodel.predict(Xtest)
print('Train R2',rdmodel.score(Xtrain,ytrain))
print('Test R2',r2_score(ytest,ypred))
print('Test RMSE',np.sqrt(mean_squared_error(ytest,ypred)))
```

```
Train R2 0.643300773530937
Test R2 0.6431036919663506
Test RMSE 5736.213796154089
```

6. Elastic Net:

Elastic net linear regression uses the penalties from both the lasso and ridge techniques to regularize regression models. The technique combines both the lasso and ridge regression methods by learning from their shortcomings to improve the regularization of statistical models.

```
# Using Elastic Net
enet = ElasticNet()
enetmodel = enet.fit(Xtrain,ytrain)
ypred = enetmodel.predict(Xtest)
print('Train R2',enetmodel.score(Xtrain,ytrain))
print('Test R2',r2_score(ytest,ypred))
print('Test RMSE',np.sqrt(mean_squared_error(ytest,ypred)))
```

```
Train R2 0.5415393499001895
Test R2 0.5417272991823014
Test RMSE 6500.043987071492
```

The model efficiency of Elastic Net is Lower than Ridge, Lasso and Linear Models.

Conclusion: All Linear Models are under-fitting. Meaning models are not able to learn and capture significant amount of the data.

8. Decision Tree:

Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented just by a discrete, known set of numbers or values.

```
# Using Decision Tree Regressor
dt = DecisionTreeRegressor()
model_dt = dt.fit(Xtrain,ytrain)
ypred = model_dt.predict(Xtest)

print('Train score',model_dt.score(Xtrain,ytrain))
print('Test R2',r2_score(ytest,ypred))
print('Test RMSE',np.sqrt(mean_squared_error(ytest,ypred)))
```

```
Train score 0.9998590115886702
Test R2 0.7885700252265521
Test RMSE 4415.069871670301
```

With training R2 of 99% and Test R2 of 78% we can see that Model is performing good on training but Test R2 is varying greatly from Training R2 .

9. Random Forest:

Random Forest Regression is **a supervised learning algorithm that uses ensemble learning method for regression**. Ensemble learning method is a technique that combines predictions from multiple machine learning algorithms to make a more accurate prediction than a single model.

```
# Using Random Forest Regressor
rf = RandomForestRegressor()
model_rf = rf.fit(Xtrain,ytrain)
ypred = model_rf.predict(Xtest)
print('Train score',model_rf.score(Xtrain,ytrain))
print('Test R2',r2_score(ytest,ypred))
print('Test RMSE',np.sqrt(mean_squared_error(ytest,ypred)))
```

```
Train score 0.9838845953588847
Test R2 0.8896809613800881
Test RMSE 3189.183703821592
```

There is 10% difference between Train and Test R2. That Means the model is still underfitting when it comes to testing. However so far Random Forest has given the best result.

Models With Best Features(Using sequential feature selection):

Feature Selection is **the method of reducing the input variable to your model by using only relevant data and getting rid of noise in data**. It is the process of automatically choosing relevant features for your machine learning model based on the type of problem you are trying to solve.

```
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
lr = LinearRegression() # Initiating Model
# Using Linear Regression as estimator.
sfs1 = sfs(estimator = lr,
           k_features='best',
           forward=True,
           scoring='neg_mean_squared_error')

sfs1.fit(X, y) #Fitting x, y

```

Best Features

```
sfs1.k_feature_names_ # Best Features For Linear Regression.
```

```
('year',
 'model',
 'cylinders',
 'title_status',
 'type',
 'odometer',
 'age',
 'fuel_electric',
 'fuel_gas',
 'fuel_hybrid',
 'transmission_manual',
 'transmission_other',
 'drive_fwd')
```

```
:  1  X2 = X.iloc[:,[0, 2, 4, 5, 6, 8, 9, 10, 11, 12, 14, 15, 16]]
```

Building Linear regression Model:

```
1  # Creating Train test and Model
2
3  Xtrain,Xtest,ytrain,ytest = train_test_split(X2,y,test_size=0.20,random_state=0)
4
5  lr = LinearRegression()
6
7  lrmodel = lr.fit(Xtrain,ytrain)
8
9  ypred = lrmodel.predict(Xtest)
10
11 print('Train R2',lrmodel.score(Xtrain,ytrain))
12 print('Test R2',r2_score(ytest,ypred))
13 print('Test RMSE',np.sqrt(mean_squared_error(ytest,ypred)))
```

```
Train R2 0.6367530732042168
Test R2 0.6372557317280241
Test RMSE 5783.018497548499
```

Result: Instead of R2 increasing it decreased but not by large margin Meaning that the insignificant variables were removed. But Linear Model is still underperforming so we will not be using it for final Modeling.

Building Models Using Ensemble Techniques:

Ensemble methods are **techniques that aim at improving the accuracy of results in models by combining multiple models instead of using a single model**. The combined models increase the accuracy of the results significantly. This has boosted the popularity of ensemble methods in machine learning.

1. ADA Boosting:

An AdaBoost regressor is a meta-estimator that begins by fitting a regressor on the original dataset and then fits additional

copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction.

```
# Building Ensemble Model
# Using ADA Boost on Decision Tree
from sklearn.ensemble import AdaBoostRegressor
adr = AdaBoostRegressor(base_estimator=DecisionTreeRegressor(),
                        n_estimators=150,random_state=0)

model_adr = adr.fit(Xtrain,ytrain)
ypred = model_adr.predict(Xtest)

print('Train score',model_adr.score(Xtrain,ytrain))
print('Test R2',r2_score(ytest,ypred))
print('Test RMSE',np.sqrt(mean_squared_error(ytest,ypred)))
```

```
Train score 0.9897256456606472
Test R2 0.8943281692310152
Test RMSE 3121.288540038411
```

2. Gradient Boosting:

This estimator builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.

```
# Using Gradient Boosting Regressor
gbr = GradientBoostingRegressor()

model_gbr = gbr.fit(Xtrain,ytrain)
ypred = model_gbr.predict(Xtest)

print('Train score',model_gbr.score(Xtrain,ytrain))
print('Test R2',r2_score(ytest,ypred))
print('Test RMSE',np.sqrt(mean_squared_error(ytest,ypred)))
```

```
Train score 0.7696596686977151
Test R2 0.7702947977828216
Test RMSE 4601.926917996675
```

3. Bagging:

A Bagging regressor is an ensemble meta-estimator that fits base regressors each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.

```
br = BaggingRegressor(base_estimator=DecisionTreeRegressor(),n_estimators=150,random_state=0)

model_br = br.fit(Xtrain,ytrain)
ypred = model_br.predict(Xtest)

print('Train score',model_br.score(Xtrain,ytrain))
print('Test R2',r2_score(ytest,ypred))
print('Test RMSE',np.sqrt(mean_squared_error(ytest,ypred)))
```

```
Train score 0.984128697502642
Test R2 0.8905151432137143
Test RMSE 3177.1032535638383
```

4. Stacking:

Stacked generalization consists in stacking the output of individual estimator and use a regressor to compute the final prediction. Stacking allows to use the strength of each individual estimator by using their output as input of a final estimator.

Note that estimators_ are fitted on the full X while final_estimator_ is trained using cross-validated predictions of the base estimators using cross_val_predict.

```
# Stacking Models
models = [('lr',LinearRegression()),
          ('gbr',GradientBoostingRegressor()),
          ('dt',DecisionTreeRegressor())]

sr = StackingRegressor(estimators = models,
                       final_estimator = RandomForestRegressor(),
                       cv = KFold(n_splits=5,shuffle=True,random_state=0))

model_sr = sr.fit(Xtrain,ytrain)
ypred = model_sr.predict(Xtest)

print('Train score',model_sr.score(Xtrain,ytrain))
print('Test R2',r2_score(ytest,ypred))
print('Test RMSE',np.sqrt(mean_squared_error(ytest,ypred)))
```

```
Train score 0.9192420555300274
Test R2 0.8427847128456796
Test RMSE 3807.1649069731875
```

Building Final Model Using K-Fold Cross Validation And Random Forest:

K-Fold:

Provides train/test indices to split data in train/test sets. Split dataset into k consecutive folds (without shuffling by default).Each fold is then used once as a validation while the k - 1 remaining folds form the training set.

```
# Cross Validation
from sklearn.model_selection import KFold

# Reseting Index
X1 = X.reset_index(drop=True)
y1 = y.reset_index(drop=True)

rf = RandomForestRegressor() # Initiating Model
# Applying Kfold with 5 splits
kfold = KFold(n_splits=5, shuffle=True,random_state=1)

pred = [] # Making empty list to save predicted values
for train_index, test_index in kfold.split(X1,y1):
    x_train = X1.loc[train_index]
    y_train = y1.loc[train_index]
    x_test = X1.loc[test_index]
    y_test = y1.loc[test_index]
    pred.append(rf.fit(x_train,y_train).predict(x_test))

finalpred = pd.DataFrame(pred).T.median(axis=1)
```

```
print('Test R2',r2_score(y_test,finalpred))
print('Test RMSE',np.sqrt(mean_squared_error(y_test,finalpred)))
```

```
Test R2 0.9762200772704244
Test RMSE 1480.6553733439669
```

As we can see RMSE has significantly Decreased and R2 Has increased by a lot.

Conclusion

Models Evaluation:

It helps to find the best model that represents our data and how well the chosen model will work in the future

Model Name	R_Squared	RMSE
Decision Tree Regressor	0.79	4389.84
Linear Regression	0.68	5399.73
Ridge	0.68	5399.73
Lasso	0.68	5399.99
Elastic Net	0.49	6888.86

Model Name	R_Squared	RMSE
Random Forest Regressor with 5 CV	0.98	1488.69
Random Forest Regressor	0.89	3188.09
Linear Regression(With Transformation)	0.64	5735.92
Linear Regression(With Scaling)	0.44	7180.49
Linear Regression(Without Scaling)	0.44	7200.01

Model Name	Training R_Squared	Test R_Squared	RMSE
Decision Tree Regressor(ADA BOOST)	0.99	0.89	3121.29
Decision Tree Regressor(Bagging Regressor)	0.98	0.89	3177.10
Random Forest Regressor	0.98	0.89	3177.21
Stacking Regressor(Final RF)	0.92	0.84	3812.65
Decision Tree Regressor	1.00	0.79	4413.90
Gradient Boosting Regressor	0.77	0.77	4601.93
Linear Regression	0.64	0.64	5736.19
Ridge	0.64	0.64	5736.21
Lasso	0.64	0.64	5736.47
Elastic Net	0.54	0.54	6500.04

The most challenging part of this project was understanding the dataset and its various features. We budgeted the majority of our time toward understanding what information each feature contained, how this information was distributed, and what sort of preprocessing techniques would need to be applied. Dealing with the dataset's overabundance of missing values, or `NaN' entries, proved to be the trickiest part of the data preprocessing phase and since most machine learning algorithms aren't designed to overlook missing entries, We had to overcome this challenge. We implemented several steps to pick the best features as much as possible.

From the overall model, we conclude that random forest with K-Fold is the best fit model. It has an R2 of 98% and RMSE 1488.69

References:

1. https://www.kaggle.com/datasets/austinreese/craigslist-carstrucks-data
2. https://github.com/jekhor/aerial-cars-dataset
3. https://github.com/AustinReese/UsedVehicleSearch
4. https://www.cii.in/sectors.aspx?enc=prvePUj2bdMtgTmvPwvisYH+5EnGjyGXO9hLECvTuNspZMG2krVmNXVq1Qz72doM

Declaration:

This is to declare that the dataset that we are using for our capstone project does not have any relevant legality associated to it and can be used to showcase the work we do on it as a presentation in Great Learning.